

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record.

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problems Mailbox.**



## PATENT ABSTRACTS OF JAPAN

(11) Publication number: **10260997 A**(43) Date of publication of application: **29.09.98**

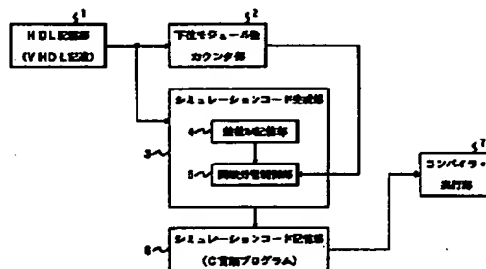
(51) Int. Cl

**G06F 17/50**(21) Application number: **09066701**(71) Applicant: **TOSHIBA CORP**(22) Date of filing: **19.03.97**(72) Inventor: **KUROSAWA YUICHI****(54) LOGIC SIMULATION SYSTEM****(57) Abstract:**

**PROBLEM TO BE SOLVED:** To provide the logic simulation system which can suppress increases in the processing time of compilation and the generating process time of object codes and actualize simulation in an adequate processing time even if an HDL (hardware description language) description quantity increases in a process for generating a program for elaboration of a compilation type HDL simulation system.

**SOLUTION:** The system which performs logic simulation by converting hardware specifications to be designed which are described in VHDL into a C-language program is equipped with a low-order module number counter part 2 which counts component instances appearing in the VHDL description in the process for generating a program for elaboration and a simulation code generation part 3 which generates a function of the program each time the counter value of the counter part 2 reaches a specific reference value M.

COPYRIGHT: (C)1998,JPO



(19)日本国特許庁 (J P)

(12) 公 開 特 許 公 報 (A)

(11)特許出願公開番号

特開平10-260997

(43)公開日 平成10年(1998)9月29日

(51)Int.Cl.<sup>6</sup>

G 0 6 F 17/50

識別記号

F I

G 0 6 F 15/60

6 6 4 K

審査請求 未請求 請求項の数10 O L (全 14 頁)

(21)出願番号

特願平9-66701

(22)出願日

平成9年(1997)3月19日

(71)出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72)発明者 黒澤 雄一

東京都青梅市末広町2丁目9番地 株式会

社東芝青梅工場内

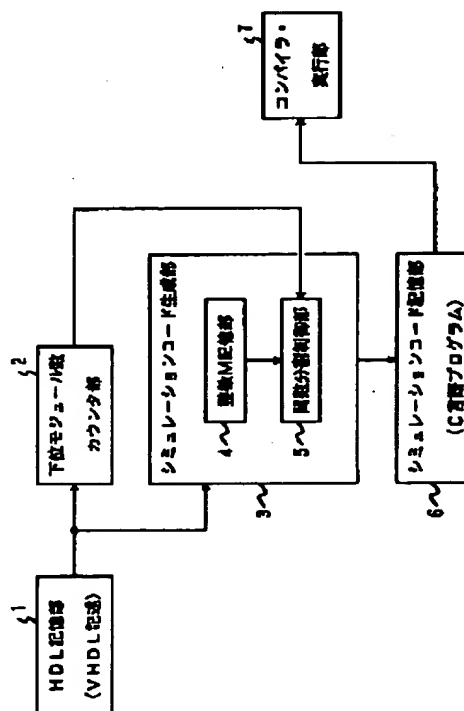
(74)代理人 弁理士 鈴江 武彦 (外6名)

(54)【発明の名称】 論理シミュレーション・システム

(57)【要約】

【課題】コンパイル方式のHDLシミュレーション・システムのエラボレーション用プログラムを生成する過程において、HDL記述量が増大した場合でも、コンパイルの処理時間及びオブジェクトコードの生成処理時間の増大化を抑制し、妥当な処理時間によるシミュレーションを実現することができる論理シミュレーション・システムを提供することにある。

【解決手段】設計対象のハードウェア仕様をVHDLにより記述された仕様記述をC言語プログラムに変換して、論理シミュレーションを実行するシステムにおいて、エラボレーション用プログラムを生成する過程において、VHDL記述中出现するコンポーネント・インスタンスの数をカウントする下位モジュール数カウンタ部2と、このカウンタ部2のカウント値が所定の基準値M毎にプログラムの関数を生成するシミュレーションコード生成部3とを備えている。



## 【特許請求の範囲】

【請求項1】 設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、

前記仕様記述からデータ構造を構築するための処理過程において、前記仕様記述中に出現する下位モジュールの数をカウントする手段と、

前記下位モジュールの数のカウント値が所定の基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記プログラムの関数を生成する手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項2】 設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムであって、

前記所定のハードウェア記述言語により記述された仕様記述を記憶する記憶手段と、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中に出現する下位モジュールの数をカウントするカウンタ手段と、

予め記憶した所定の基準値と前記カウンタ手段のカウント値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記プログラムの関数を生成し、この複数の関数からなる前記プログラムのソースコードを生成するコード生成手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項3】 前記所定のプログラミング言語はC言語であり、

前記C言語プログラムをコンパイルするコンパイラと、コンパイルされたコードとライブラリとのリンク処理を実行するリンカと、

リンク処理された後のオブジェクトコードを実行することにより、前記仕様記述のシミュレーションを実行するための実行手段とを具備したことを特徴とする請求項1または請求項2記載の論理シミュレーション・システム。

【請求項4】 設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムに適用するシミュレーション方法であって、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前

記仕様記述中に出現する下位モジュールの数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウント値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記プログラムの関数を生成する処理と、生成された複数の関数からなる前記プログラムのソースコードを生成する処理とからなることを特徴とするシミュレーション方法。

10 【請求項5】 コンピュータシステムにより読取可能な記録媒体であり、

設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、Cプログラミング言語によるプログラムコードに変換して、このプログラムコードにより論理シミュレーションを実行するためのプログラムを格納した記録媒体であって、

前記プログラムは、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中に出現する下位モジュールの数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウント値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記下位モジュールの数を単位として前記Cプログラミング言語における関数を生成する処理と、

生成された複数の関数からなる前記Cプログラミング言語のソースコードを生成する処理と、

30 前記ソースコードをコンパイルし、かつライブラリとのリンクする処理とを有することを特徴とする記録媒体。

【請求項6】 設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、

前記仕様記述からデータ構造を構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントする手段と、

前記信号の数のカウント値が所定の基準値以上のとき

40 に、前記基準値に相当する前記信号の数を単位として前記プログラムの関数を生成する手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項7】 設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムであって、

前記所定のハードウェア記述言語により記述された仕様記述を記憶する記憶手段と、

50 前記仕様記述からデータ構造を前記コンピュータシステ

## 3

ムの記憶空間上に構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントするカウンタ手段と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記信号の数を単位として前記プログラムの関数を生成し、この複数の関数からなる前記プログラムのソースコードを生成するコード生成手段とを具備したことを特徴とする論理シミュレーション・システム。

【請求項8】 前記所定のプログラミング言語はC言語であり、

前記C言語プログラムをコンパイルするコンパイラと、コンパイルされたコードとライブラリとのリンク処理を実行するリンカと、

リンク処理された後のオブジェクトコードを実行することにより、前記仕様記述のシミュレーションを実行するための実行手段とを具備したことを特徴とする請求項6または請求項7記載の論理シミュレーション・システム。

【請求項9】 設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムをコンピュータシステムにより実行する論理シミュレーション・システムに適用するシミュレーション方法であって、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウンタ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記信号の数を単位として前記プログラムの関数を生成する処理と、

生成された複数の関数からなる前記プログラムのソースコードを生成する処理とからなることを特徴とするシミュレーション方法。

【請求項10】 コンピュータシステムにより読取可能な記録媒体であり、

設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、Cプログラミング言語によるプログラムコードに変換して、このプログラムコードにより論理シミュレーションを実行するためのプログラムを格納した記録媒体であって、

前記プログラムは、

前記仕様記述からデータ構造を前記コンピュータシステムの記憶空間上に構築するための処理過程において、前記仕様記述中において宣言された信号の数をカウントする処理と、

予め記憶した所定の基準値と前記カウンタ手段のカウ

## 4

タ値とを比較して、前記カウンタ値がその基準値以上のときに、前記基準値に相当する前記信号の数を単位として前記Cプログラミング言語における関数を生成する処理と、

生成された複数の関数からなる前記Cプログラミング言語のソースコードを生成する処理と、

前記ソースコードをコンパイルし、かつライブラリとのリンクする処理とを有することを特徴とする記録媒体。

【発明の詳細な説明】

10 【0001】

【発明の属する技術分野】本発明は、コンピュータを使用して電子回路などのハードウェアの設計を支援するためのCADシステムに適用し、特にハードウェア記述言語により記述された論理回路設計の仕様記述の検証を行なうための論理シミュレーション・システムに関する。

【0002】

【従来の技術】近年、LSIなどの集積回路の設計は、設計対象のハードウェア（具体的には論理回路）の複雑化及び大規模化に伴って、ハードウェアの設計仕様をハードウェア記述言語（HDL）により記述し、CADシステム（コンピュータシステム）を利用して論理回路の設計や設計検証を行なう設計手法が定着しつつある。このようなHDLによる仕様記述を設計検証するためのシステムは、論理シミュレーション・システムまたは論理シミュレータ（HDLシミュレータ）と呼ばれている。HDLとしては、世界標準的にはVHDL（VHSIC hardware description language）またはVerilogHDLが広く使用されている。後述する図2のHDL記述は、VHDL記述の具体例である。

30

【0003】論理シミュレーション・システムは、通常ではHDL記述をC言語などのプログラミング言語からなるプログラムに変換し、このプログラムをコンパイルしてオブジェクトコードを生成し、このオブジェクトコードを実行する機能からなる。この方式はコンパイル方式と呼ばれている。

【0004】シミュレーションのプロセスは、HDL記述中に出現する信号、変数、モジュールなどのデータ構造をコンピュータの記憶空間上に構築する処理であるエラボレーション（elaboration）及びこのデータ構造をアクセスしながらHDL記述中のステートメントを実行する処理に大別される。また、C言語プログラムに変換する場合も、このプログラムをエラボレーション用プログラム及びステートメント実行用プログラムに分けて生成するのが一般的である。

【0005】エラボレーション用プログラムは、HDL記述の階層（モジュール）毎にプログラミング言語の1関数を生成し、またステートメント実行用プログラムは例えばVHDLやVerilogHDLのプロセスやサブプログラム毎にプログラミング言語の1関数を生成す

50

るのが一般的である。

【0006】ところで、例えばC言語プログラムにおいて、プログラムを構成する1関数のサイズが一定以上になった場合に、その関数を機能的に複数の関数に分割した方がコンパイルしたときのコンパイル時間を短縮できるという性質がある。ここで、1つの階層（モジュール）のHDL記述が大規模になった場合でも、ステートメント実行用プログラムは、プロセスやサブプログラム毎に1関数を生成できるため、生成される個々の関数は比較的大規模にならずにすむ。しかし、エラボレーション用プログラムは、特にゲートアレイやスタンダードセルを使用したネットリストのように信号数やサブモジュール数（ゲートアレイやスタンダードセルのインスタンス数に相当。）が多くなる場合に、生成される1関数が大きくなり、関数のコンパイルに要する時間が増大化、あるいはコンパイルそのものができなくなるという問題がある。

【0007】図6は、従来手法において、図2に示すVHDL記述に対するエラボレーション用コードのC言語による具体例を示す図である。図6において、10～13行目、15～18行目、20～23行目、25～28行目がそれぞれコンポーネント・インスタンス文C1、C2、C3、C4のエラボレーション用コード（以下エラボレーション・コードと称する）に相当する。この手法では、コンポーネント・インスタンス文C1、C2、C3、C4全てのエラボレーション・コードを同一の関数new\_lib1\_ex1\_structure内に出力している。このため、VHDL記述中にn個のコンポーネント・インスタンス文が含まれる場合には、生成されるエラボレーション・コードのステートメント数は「3\*n」以上になる。例えば10000個のセルからなるゲートアレイ回路のエラボレーション・コードは、30000ステートメント以上になる。従って、コンポーネント・インスタンス文の個数nが非常に大きくなると、それに伴ってエラボレーション・コードのコンパイル時間が増大し、また最悪の場合にはコンパイル不能になる可能性がある。

#### 【0008】

【発明が解決しようとする課題】前述したように、HDL記述の階層（モジュール）毎に、C言語などのエラボレーション用プログラムを生成する従来方式では、1階層の記述量が増大した場合に、それに伴ってエラボレーション・コードのコンパイル時間が増大化し、オブジェクトコードの生成の処理時間が非常に増大化したり、またコンパイル自体が不可能になるという問題がある。

【0009】そこで、本発明の目的は、いわゆるコンパイル方式のHDLシミュレーション・システムのエラボレーション用プログラムを生成する過程において、HDL記述量が増大した場合でも、コンパイルの処理時間及びオブジェクトコードの生成処理時間の増大化を抑制

し、妥当な処理時間によるシミュレーションを実現することができる論理シミュレーション・システムを提供することにある。

#### 【0010】

【課題を解決するための手段】本発明は、設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、仕様記述からデータ構造を構築するための処理過程において仕様記述中に出現する下位モジュールの数をカウントする手段と、下位モジュールの数のカウント値が所定の基準値以上のときに基準値に相当する前記下位モジュールの数を単位としてプログラムの関数を生成する手段とを備えたシステムである。

【0011】即ち、本システムは、HDL記述から例えばC言語プログラムを生成するときのエラボレーション用プログラムの生成過程において、1階層（1モジュール）の記述量の規模（カウント値）が大きい場合に、複数の関数に分割してプログラムを生成する方式である。従って、HDL仕様記述に含まれる下位モジュール数がM個以上である場合に、M個の下位モジュールを単位としてプログラムの1関数を生成するため、1関数のサイズの増大化を抑制できる。これにより、コンパイルの処理及びオブジェクトコードの生成処理を妥当な処理時間により実行することができる。

【0012】また、本発明は、設計対象のハードウェア仕様を所定のハードウェア記述言語により記述された仕様記述を、所定のプログラミング言語によるプログラムに変換して、このプログラムを実行する論理シミュレーション・システムであって、仕様記述からデータ構造を構築するための処理過程において、仕様記述中において宣言された信号の数をカウントする手段と、信号の数のカウント値が所定の基準値以上のときに、基準値に相当する信号の数を単位としてプログラムの関数を生成する手段とを備えたシステムである。

【0013】即ち、本システムは、HDL記述から例えばC言語プログラムを生成するときのエラボレーション用プログラムの生成過程において、仕様記述に含まれる信号数が多い場合に、複数の関数に分割してプログラムを生成する方式である。従って、HDL仕様記述中に宣言された信号数がN個以上である場合に、N個の信号を単位としてプログラムの1関数を生成するため、1関数のサイズの増大化を抑制できる。これにより、コンパイルの処理及びオブジェクトコードの生成処理を妥当な処理時間により実行することができる。

#### 【0014】

【発明の実施の形態】以下図面を参照して本発明の実施の形態を説明する。

（第1の実施形態）図1は本実施形態に係るシステ

10

20

30

40

50

ム構成を示すブロック図であり、図2はVHDL記述の具体例を示す図であり、図3は本実施形態の動作を説明するためのフローチャートである。

【0015】本実施形態は、図2に示すように、設計対象の論理回路をVHDL記述により仕様記述を行なう場合について適用し、この仕様記述をC言語プログラムに変換してシミュレーションを実行するシステムを想定する。なお、VHDLに関する文法、用語などについては、例えば「VHDLハードウェア記述言語 日本規格協会 ISBN4-5425-40136-7」などの文献に記載されている。

(システム構成) 本システムは、図1に示すように、HDL記憶部1と、下位モジュール数カウンタ部2と、シミュレーションコード生成部3と、シミュレーションコード記憶部6と、コンパイラ・実行部7とを有する。シミュレーションコード生成部3は、整数M記憶部4と関数分割制御部5とを有する。

【0016】HDL記憶部1は、VHDLにより記述されたソースプログラム（以下VHDL記述と称する）を記憶するための要素である。ここで、VHDL記述は、図2に示すように、「entity ex1」及びそのstructureという名前のアーキテクチャ（architecture）の記述である（図2の1、5行目を参照）。

【0017】下位モジュール数カウンタ部2は、HDL記憶部1に記憶されたVHDL記述においてコンポーネント・インスタンス文を1個ずつ取り出して、その数をカウントするための要素である。コンポーネント・ポートインスタンス文は、VHDLにおいて下位モジュールとの接続関係を表す構文であり、例えば図2の34、42、51、58行目の文が相当する。例えば、34行目のコンポーネント・インスタンス文は識別子がC1であり、6行目で宣言された下位モジュールSEL2が使用されている。

【0018】シミュレーションコード生成部3は、前述したように、整数M記憶部4と関数分割制御部5とを有し、HDL記憶部1に記憶されたHDL記述をシミュレーションするためのC言語プログラム（シミュレーションコード）を生成するための本実施形態の主要素である。整数M記憶部4は、コンポーネント・インスタンス文に対するエラボレーション・コードの出力を制御するための整数M（基準値）を記憶する。関数分割制御部5は、下位モジュール数カウンタ部2によりカウントされたコンポーネント・インスタンス数（下位モジュール数）がM個になる毎に、新しいC関数（C言語プログラムの関数）を生成し、当該コンポーネント・インスタンス文のエラボレーション・コードをそのC関数中に出力する処理を行なう。

【0019】シミュレーションコード記憶部6は、シミュレーションコード生成部3により生成されたC言語プ

ログラムを記憶する。なお、シミュレーションコード生成部3は、VHDL記述のエラボレーション・コードとステートメントを実行するためのコードとを生成する。本発明はエラボレーション・コードを生成する処理に関するものであるため、本実施形態ではエラボレーション・コードの生成処理のみについて説明する。エラボレーション・コードは、前述したように、VHDL記述中に出現する信号、変数、モジュールなどのデータ構造をコンピュータの記憶空間上に構築するための処理を実行するためのコードである。

（第1の実施形態のエラボレーション・コードの生成処理）本実施形態では、HDL記憶部1には、図2に示すようなVHDL記述が記憶されている。また、このVHDL記述が含まれているライブラリ名は例えば「lib1」とであると想定する。以下、図3のフローチャートを参照して、本実施形態の処理を説明する。

【0020】まず、下位モジュールカウンタ部2がクリアされて（カウント値j=0）、関数分割制御部5の制御基準値として整数であるパラメータM（M=2とする）が整数M記憶部4にセットされる（ステップS1）。

【0021】ここで、1つのコンポーネント・インスタンス文に対しては、コンポーネント・インスタンス文以下の形式のC関数コールを生成するものとする。コンポーネント・インスタンス文以下の形式は、便宜的にスカラータイプのポートすなわち1ビット幅のポートのみからなるコンポーネントのものであると想定する。具体例としては、以下の3ステートメントを示す。

【0022】h\_comp=new\_vblk0;  
bindeda=bind\_ライブラリ名\_エンティティ名\_アーキテクチャ名\_コンポーネントインスタンス名(h\_comp, ...);  
st\_component\_attr\_func(h\_comp, "コンポーネントインスタンス名", bindeda, h0);

前記1行目の関数コールはコンポーネント・インスタンスに対応するデータ構造h\_compを生成する。2行目の関数コールの第2引数以降は、当該コンポーネントのポートに対応して、そのポートに接続されている信号のアドレスを実引数として設定する。VHDLアーキテクチャ記述内のn番目（n=0, 1, 2...）に宣言された信号のアドレスは、最初の信号（図2ではS0）のアドレスintsig0を基準として、「intsig0+n」とする。このように、1つのコンポーネント・インスタンス文に対して、上記3ステートメントを生成するが、これらのステートメントを全て同じ関数内に出力するのではなく、M個のコンポーネントインスタンス文毎に別のC関数内に出力するのが本発明の要旨である。

【0023】シミュレーションコード生成部3は、HDL記憶部1からVHDL記述中のコンポーネント・イン

スタンス文C<sub>n</sub> (n=1, 2, 3, 4) を1つ取り出す (ステップS2)。ここでは、図2に示す34行目のインスタンス文C1が得られる (ステップS3のYES)。シミュレーションコード生成部3は、カウンタ部2からのカウント値jを入力して、パラメータMとの比較処理を実行する。この比較処理により、カウント値j (ここではj=0) がMの倍数であるため、シミュレーションコード生成部3はインスタンス文C1に対する前記3ステートメントを出力するための関数gを新たに生成する (ステップS5)。この「g」の関数名は、new\_ライブラリ名\_エンティティ名\_アーキテクチャ名\_comp識別子とする。識別子にはカウンタjの値を使用する。

【0024】そして、シミュレーションコード生成部3は、生成した関数gを呼び出すコードをアーキテクチャstructureのエラボレーション関数内に出力する (ステップS6)。ここで、アーキテクチャのエラボレーション関数の名前は、new\_ライブラリ名\_エンティティ名\_アーキテクチャ名とし、上位階層に対するデータ構造へのポインタparentを引数にもつものとする。

【0025】次に、シミュレーションコード生成部3は、インスタンス文C1に対する3ステートメントを関数g、即ちnew\_lib1\_ex1\_structure\_comp0の中に出力する (ステップS7)。そして、下位モジュールカウンタ部2がインクリメントして (カウント値j=1)、シミュレーションコード生成部3は、HDL記憶部1からVHDL記述中の次のコンポーネント・インスタンス文C2を取り出す (ステップS8, S2へリターン)。ここでは、図2に示す42行目のインスタンス文C2が得られる (ステップS3のYES)。シミュレーションコード生成部3は、カウンタ部2からのカウント値jを入力して、パラメータMとの比較処理を実行する。この比較処理により、カウント値j (ここではj=1) がMの倍数ではないため、シミュレーションコード生成部3はインスタンス文C2のエラボレーション・コードをインスタンス文C1のエラボレーション・コードの出力先と同じ関数g、即ちnew\_lib1\_ex1\_structure\_comp0の中に出力する (ステップ4のNO, S7)。

【0026】そして、下位モジュールカウンタ部2がインクリメントして (カウント値j=2)、シミュレーションコード生成部3は、HDL記憶部1からVHDL記述中の次のコンポーネント・インスタンス文C3を取り出す (ステップS8, S2へリターン)。以下同様の処理を繰り返す。即ち、図2に示す51行目のインスタンス文C3が得られる (ステップS3のYES)。j (=2) がM (=2) の倍数であるため、コンポーネント・インスタンス文C1のときと同様に、ステップS5の処理に移行する。即ち、新しい関数new\_lib1\_e

xl\_structure\_comp2を生成する。さらに、これを呼び出すコードをアーキテクチャstructureのエラボレーション関数new\_lib1\_ex1\_structureの中に出力する (ステップS6)。そして、HDL記憶部1からVHDL記述中の次のコンポーネント・インスタンス文C4を取り出す (ステップS8, S2へリターン)。以下同様の処理を繰り返す。即ち、図2に示す58行目のインスタンス文C4が得られる (ステップS3のYES)。j (=3) がM (=2) の倍数ではないため、コンポーネント・インスタンス文C2のときと同様に、ステップS7の処理に移行する。即ち、インスタンス文C4に対するエラボレーション・コードを関数new\_lib1\_ex1\_structure\_comp2の中に出力し、カウンタ部2がインクリメントする (ステップS7, S8)。

【0027】シミュレーションコード生成部3は、HDL記憶部1からVHDL記述中の次のコンポーネント・インスタンス文C4を取り出す (ステップS8, S2へリターン)。以下同様の処理を繰り返す。即ち、図2に示す58行目のインスタンス文C4が得られる (ステップS3のYES)。j (=3) がM (=2) の倍数ではないため、コンポーネント・インスタンス文C2のときと同様に、ステップS7の処理に移行する。即ち、インスタンス文C4に対するエラボレーション・コードを関数new\_lib1\_ex1\_structure\_comp2の中に出力し、カウンタ部2がインクリメントする (ステップS7, S8) 次に、HDL記憶部1には他にコンポーネント・インスタンス文は存在しないため、シミュレーションコード生成部3は処理を終了する (ステップS3のNO)。この結果、シミュレーションコード記憶部6には、図4に示すように、生成されたアーキテクチャstructureのエラボレーション・コード (一部) が格納される。また、ステップS4の処理で生成されたコンポーネント・インスタンスM (=2) 個分のエラボレーション・コードが格納される (図5を参照)。ここで、図4の関数中では、図5の関数をコールしている。図5の最初の関数new\_lib1\_ex1\_structure\_comp0はコンポーネント・インスタンスC1, C2のエラボレーションを行うコードであり、2個目の関数new\_lib1\_ex1\_structure\_comp2はコンポーネント・インスタンスC3, C4のエラボレーションを行うコードである。

【0028】ところで、前述したように、図6は従来方式により、図2のVHDL記述から生成したエラボレーション・コード (一部) を示す。図6では、コンポーネントインスタンス文C1, C2, C3, C4全てのエラボレーション・コードが同一の関数new\_lib1\_ex1\_structure内に含まれる。このため、VHDL記述にn個のコンポーネント・インスタンス文が含まれる場合、生成されるエラボレーション・コードのステートメント数は3\*n以上になる。このため、nが非常に大きくなるとエラボレーション・コードがコンパイルできなくなる可能性がある。

【0029】これに対して、本実施形態の方式であれば、コンポーネント・インスタンス文のエラボレーション・コードを全て同一の関数内に出力するのではなく、



M個毎に異なる関数内に出力するため、コンポーネント・インスタンス文C<sub>n</sub>の数がいくつであっても、コンポーネント・インスタンス文のエラボレーション・コードを含む個々の関数のサイズは一定以下(3\*Mステートメント程度)になる。換言すれば、VHDL記述に含まれるコンポーネント・インスタンス文(HDL記述の下位モジュール)をカウントし、所定値M(例えば2)毎に関数g(図5の3行目と26行目)を生成し、この関数gからなるC言語プログラムを生成する。従って、コンパイラ・実行部7が生成されたC言語プログラムをコンパイルするときに、各関数のサイズに制限されているため、コンパイル処理時間が極端に増大したり、またはコンパイル不能になるようなことはなく、妥当なコンパイル処理時間でオブジェクトコードを生成することが可能となる。

(第2の実施形態)図7は第2の実施形態に関する論理シミュレーション・システムの要部を示すブロック図である。第2の実施形態のシステムは、前述の第1の実施形態において下位モジュール数カウンタ部2の代わりに信号数カウンタ部70を有し、さらに整数M記憶部4の代わりにN記憶部72を有するものである。信号数カウンタ部70は、HDL記憶部1に記憶されたVHDL記述中において宣言された信号数をカウントする。また、N記憶部72は、コンポーネント・インスタンス文に対するエラボレーション・コードの出力先に制御するためのパラメータN(基準値の整数)を記憶するための要素である。ここで、図2に示すVHDL記述において、23~31行目のキーワード「signal」が信号の宣言部分である。

【0030】他の要素は第1の実施形態のシステムと同様の機能を有する。即ち、シミュレーションコード生成部71は、N記憶部72と関数分割制御部73を有し、HDL記憶部1に記憶されたVHDL記述をシミュレーションするためのC言語プログラムを生成する。関数分割制御部73は、信号数カウンタ部70によりカウントされた信号数がN個になる毎に、新しいC関数を生成し、当該信号のエラボレーション・コードをそのC関数中に出力する。シミュレーション・コード記憶部6は、シミュレーションコード生成部71により生成されたC言語プログラムを記憶する。

(第2の実施形態のエラボレーション・コードの生成処理)本実施形態においても、第1の実施形態と同様に、エラボレーション・コードの生成処理のみについて説明し、VHDL記述が含まれているライブラリ名は「lib1」とであると想定する。以下、図8のフローチャートを参照して、本実施形態の処理を説明する。

【0031】まず、信号数カウンタ部70がクリアされて(カウント値i=0)、関数分割制御部73の制御基準値として整数であるパラメータN(N=6とする)がN記憶部72にセットされる(ステップS10)。

【0032】ここで、1つの信号宣言に対しては、以下の形式のC関数コールを生成するものとする。以下の形式は、便宜的にスカラータイプの信号のものであると想定する。具体例としては、以下の1ステートメントを示す。

【0033】`st_signal_attr_func(intaig0+n, SCALAR_SIGNAL, NULL, h0);`

この関数コールは、信号に対するデータ構造に属性をセットするものである。第1引数は信号のアドレスを設定する。VHDLアーキテクチャ記述内のn番目(n=0, 1, 2...)に宣言された信号のアドレスは、最初の信号(図2ではS0)のアドレスintaig0を基準として、intaig0+nとする。第2引数は、信号がスカラータイプ(1ビットの信号)であることを示すコードである。第3引数は信号がベクトルのときのみ使用する情報であり、スカラー信号ではNULLとする。最後の引数は、信号が定義された階層構造へのポインタをセットする。このように、1つの信号宣言に対して、上記1ステートメントを生成するが、これらのステートメントを全て同じ関数内に出力するのではなく、N個の信号毎に別のC関数内に出力するのが本発明の趣旨である。

【0034】シミュレーションコード生成部71は、HDL記憶部1からVHDL記述中に宣言された信号S<sub>n</sub>を1つ取り出す(ステップS11)。ここでは、図2に示す23行目のS0が得られる(ステップS12のYES)。シミュレーションコード生成部3は、カウンタ部70からのカウント値iを入力して、パラメータNとの比較処理を実行する。この比較処理により、カウント値I(ここではi=0)がNの倍数であるため、シミュレーションコード生成部71は信号S0に対する前記1ステートメントを出力するための関数fを新たに生成する(ステップS14)。この「f」の関数名は、new\_ライブラリ名\_エンティティ名\_アーキテクチャ名\_subsig識別子とし、識別子には、カウンタiの値を使用する。

【0035】そして、シミュレーションコード生成部71は、生成した関数fを呼び出すコードをアーキテクチャstructureのエラボレーション関数内に出力する(ステップS15)。ここで、アーキテクチャのエラボレーション関数は、前述の第1の実施形態と同様である。

【0036】次に、シミュレーションコード生成部71は、信号S0に対するステートメントを関数f、即ちnew\_lib1\_ex1\_structure\_subsig0の中出力する(ステップS16)。そして、カウンタ部70がインクリメントして(カウント値i=1)、シミュレーションコード生成部71は、HDL記憶部1からVHDL記述中の次の信号S1(23行目)

を取り出す(ステップS17, S11へリターン)。シミュレーションコード生成部71は、カウンタ部70からのカウント値*i*を入力して、パラメータ*N*との比較処理を実行する。この比較処理により、カウント値*i*(ここでは*i*=1)が*N*の倍数ではないため、シミュレーションコード生成部71は信号S1のエラボレーション・コードを信号S0のエラボレーション・コードの出力先と同じ関数*f*、即ちnew\_libl\_exl\_structure\_subsig0の中に出力する(ステップ13のNO, S16)。

【0037】そして、カウンタ部70がインクリメントして(カウント値*i*=2)、シミュレーションコード生成部71は、HDL記憶部1からVHDL記述中の次の信号を取り出す(ステップS17, S11へリターン)。以下同様の処理を繰り返す。即ち、ステップS11では、信号D0, D1, D2, D3がそれぞれ、カウント値*i*=2, 3, 4, 5のときに取り出される。これらの場合、*i*が*N*の倍数でないため、前述の信号S1の場合と同様に、そのエラボレーション・コードは関数new\_libl\_exl\_structure\_subsig0の中に出力される(ステップS16)。

【0038】次に、カウンタ部70がインクリメントして(カウント値*i*=6)、シミュレーションコード生成部71は、HDL記憶部1からVHDL記述中の次の信号Cを取り出す(ステップS17, S11へリターン)。ここで、*i*が*N*の倍数であるため、ステップS14の処理に移行する。即ち、シミュレーションコード生成部71は、新しい関数new\_libl\_exl\_structure\_subsig6を生成し、この関数をコールするコードをアーキテクチャエラボレーション関数new\_libl\_exl\_structure内に出力する(ステップS15)。以下、ステップS17において*i*=7, 8, 9となったときに、ステップS11において、それぞれ信号CLK, Q1, Q2が取り出されてゆくが、*i*が*N*の倍数でないため、これらのエラボレーション・コードは、関数new\_libl\_exl\_structure\_subsig6中に出力される(ステップS16)。そして、ステップS11において、これ以上信号が得られないため、ステップS12がNOとなって処理が終了する。

【0039】この結果、シミュレーションコード記憶部6には、図9に示すように、生成されたアーキテクチャstructureのエラボレーション・コード(一部)が格納される。また、ステップS14の処理で生成された信号*N*(=6)個分のエラボレーション・コードが格納される(図10を参照)。ここで、図9の関数中の9, 10行目では、図10の関数がコールされている。図10において、最初の関数内の7~12行目は、図2の信号S0~D3に対するエラボレーション・コードであり、2個目の関数内の22~25行目は信号C~

Q1に対するエラボレーション・コードである。

【0040】ところで、前述したように、図11は従来方式により、図2のVHDL記述から生成したエラボレーション・コード(一部)を示す。図11では、図2の記述中で宣言されたS0~Q1全ての信号10個のエラボレーション・コードが同一の関数new\_libl\_exl\_structure内に含まれる。このため、VHDL記述に*n*個の信号宣言が含まれる場合、生成されるエラボレーション・コードのステートメント数は*n*以上になる。このため、*n*が非常に大きくなるとエラボレーション・コードがコンパイルできなくなる可能性がある。

【0041】これに対して、本実施形態の方式であれば、信号のエラボレーション・コードを全て同一の関数内に出力するのではなく、*N*個ずつ異なる関数内に出力するため、信号数がいくつであっても、信号のエラボレーション・コードを含む個々の関数のサイズは一定以下(*N*ステートメント程度)になる。従って、生成されたC言語プログラムのコンパイル処理時間を短縮することができる。また、VHDL記述に含まれる信号数がいくつになってもコンパイルができなくなるということはない。従って、コンパイラ・実行部7が生成されたC言語プログラムをコンパイルするときに、各関数のサイズに制限されているため、コンパイル処理時間が極端に増大したり、またはコンパイル不能になるようなことはなく、妥当なコンパイル処理時間でオブジェクトコードを生成することが可能となる。

(第3の実施形態) 第3の実施形態は、前述の第1と第2の実施形態の組み合わせたものである。即ち、第1の実施形態は、下位モジュール(VHDL記述ではコンポーネント・インスタンス文により記述)数の多いHDL記述に対して有効であり、第2の実施形態は信号数の多いHDL記述に対して有効である。

【0042】図12は第3の実施形態に係るシステムのブロック図である。本システムにおいて、HDL記憶部1は、第1と第2の実施形態と同様に、VHDL記述を記憶する。下位モジュール数カウンタ部2は、第1の実施形態と同様に、HDL記憶部1に記憶されたVHDL記述においてコンポーネント・インスタンス文を1個ずつ取り出しその数をカウントする。信号数カウンタ部70は、第2の実施形態70と同様に、HDL記憶部1に記憶されたVHDL記述において宣言された信号を1個ずつ取り出し、その数をカウントする。

【0043】さらに、シミュレーションコード生成部80は、M記憶部4と、N記憶部72と、関数分割制御部81とを有し、HDL記憶部1に記憶されたVHDL記述をシミュレーションするためのC言語プログラムを生成する。M記憶部4は、第1の実施形態と同様に、コンポーネント・インスタンス文に対するエラボレーション・コードの出力先を制御するためのパラメータMを記憶

する。N記憶部72は、第2の実施形態と同様に、信号に対するエラボレーション・コードの出力先を制御するためのパラメータNを記憶する。関数分割制御部81は、信号に対するエラボレーション・コードを生成する場合においては、図8のフローチャートで示す同様の処理を実行し、信号数カウンタ部70によりカウントされた信号数がN個になる毎に新しいC関数を生成し、当該信号のエラボレーション・コードをそのC関数中に出力する。また、関数分割制御部81は、コンポーネント・インスタンス文に対するエラボレーション・コードを生成する場合においては、図3のフローチャートで示す同様の処理を実行し、下位モジュールカウンタ部2によりカウントされたコンポーネント・インスタンス数がM個になる毎に、新しいC関数を生成し、当該コンポーネント・インスタンスのエラボレーション・コードをそのC関数中に出力する。シミュレーションコード記憶部6は、シミュレーションコード生成部80により生成されたC言語プログラムを記憶する。

【0044】ここで、M=2、N=6として、図2のVHDL記述から、シミュレーションコード生成部80により生成されたエラボレーション・コードの一部を図13に示す。図13において、9、10行目は信号のエラボレーション・コードをコールする部分であり、15、16行目はコンポーネント・インスタンスのエラボレーション・コードをコールする部分である。コールされた信号N個分のエラボレーション・コード及びコンポーネント・インスタンスM個分のエラボレーション・コードは、それぞれ図10、図5と同様である。

（第3の実施形態の変形例1）第3の実施形態において、シミュレーションコード記憶部6に記憶される関数をそれぞれ別のファイルに記憶してもよい。別ファイルに記憶し、ファイル単位でコンパイルをかけることによって、全ての関数を1ファイルに出力する場合と比較して処理時間を短縮化できる可能性がある。

（第3の実施形態の変形例2）第3の実施形態において、生成されるエラボレーション・コードの量を制御するパラメータM、Nの設定を固定とせず、本システムが実行される前に予め外部から設定できるようにしてもよい。例えば、パラメータM、Nの値をファイルから読み込むことが考えられる。あるいは、本システムが実行されるOS（例えば、UNIX）の環境変数として定義し、設定してもよい。

（第1から第3の実施形態の変形例1）第1、第2、第3の各実施形態においては、HDL記述をC言語プログラムに変換してシミュレーションを行う方式の場合を説明したが、他のプログラミング言語に変換してシミュレーションを行う方式の場合も本発明を適用することができる。また、C言語などの高級プログラミング言語を介さず、アセンブラ言語のコードを直接生成して、シミュレーションを行う方式の場合にも適用することができ

る。アセンブラ言語のコードを生成する場合も、高級プログラミング言語の場合と同様に、エラボレーション・コードを適当なサイズに関数化して出力することによって、オブジェクトコードの生成に要する処理時間を短縮化することができる。

（第1から第3の実施形態の変形例2）第1、第2、第3の各実施形態においては、HDL記述としてVHDLの場合を示したが、VerilogHDLなど他のHDL記述の場合も適用することができる。

（第1から第3の実施形態の変形例3）第1、第2、第3の各実施形態において、エラボレーション・コードの生成処理及びコンパイラの処理を含むソフトウェアを、例えばCD-ROMなどの光ディスクやフロッピーディスクなどの磁気ディスクに格納した記録媒体も本発明の適用範囲である。この記録媒体を、パーソナルコンピュータやワークステーションなどのコンピュータシステムにセットすることにより、前述した第1、第2、第3の各実施形態における論理シミュレーションを実行することができる。

【0045】

【発明の効果】以上詳述したように本発明によれば、コンパイル方式のHDLシミュレーション・システムにおいて、特にエラボレーション用プログラムを生成する過程において、HDL記述量が増大化した場合でも、コンパイルの処理時間及びオブジェクトコードの生成処理時間の増大化を抑制することができる。従って、大規模なHDL記述に対しても妥当な処理時間によるシミュレーションを実現することができる。

【図面の簡単な説明】

【図1】本発明の第1の実施形態に関する論理シミュレーション・システムの要部を示すブロック図。

【図2】本実施形態に関するVHDL記述の具体例を示す図。

【図3】本実施形態の動作を説明するためのフローチャート。

【図4】本実施形態に関するエラボレーション・コードを示す図。

【図5】本実施形態に関するエラボレーション・コードを示す図。

【図6】従来のVHDL記述に対するエラボレーション用コードのC言語による具体例を示す図。

【図7】第2の実施形態に関する論理シミュレーション・システムの要部を示すブロック図。

【図8】第2の実施形態の動作を説明するためのフローチャート。

【図9】第2の実施形態に関するエラボレーション・コードを示す図。

【図10】第2の実施形態に関するエラボレーション・コードを示す図。

【図11】従来方式におけるエラボレーション・コード

を示す図。

【図12】第3の実施形態に関する論理シミュレーション・システムの要部を示すブロック図。

【図13】第3の実施形態に関するエラーレーション・コードを示す図。

【符号の説明】

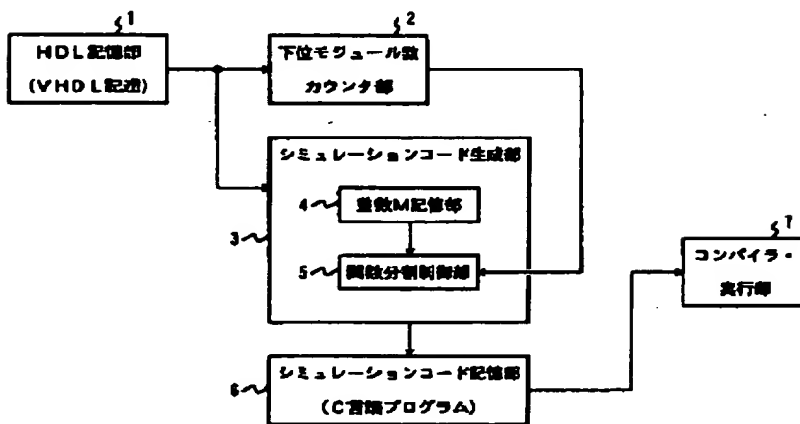
- 1…HDL記憶部
- 2…下位モジュール数カウンタ部
- 3…シミュレーションコード生成部
- 4…整数M記憶部

- 5…関数分割制御部
- 6…シミュレーションコード記憶部
- 7…コンパイラ・実行部
- 70…信号数カウンタ部
- 71…シミュレーションコード生成部
- 72…N記憶部
- 73…関数分割制御部
- 80…シミュレーションコード生成部
- 81…関数分割制御部

10

【図1】

【図4】



```

行
4  VBLKP new_libl_exl_structure(parent)
5  VBLKP parent;
   |
   |  途中省略。
   |
10  new_libl_exl_structure_comp0(h0, intsig0);
12  new_libl_exl_structure_comp2(h0, intsig0);
   |
   |  return(h0);
   |

```

【図2】

【図6】

```

行
1  entity exl is
2  end EX1;

5  architecture structure of EX1 is
   component SEL2
   port(
       I0 : in bit;
       I1 : in bit;
       C : in bit;
       O0 : out bit;
   );
   end component;

   component DFF
   port(
       D : in bit;
       CK : in bit;
       Q : out bit;
   );
   end component;

23  signal S0,S1 : bit;
   signal D0 : bit;
   signal D1 : bit;
   signal O2 : bit;
   signal O3 : bit;
   signal C : bit;
   signal CLK : bit;
   signal O0 : bit;
   signal O1 : bit;

31

```

```

行
34  begin
   C1 : SEL2
   port map(
       I0 => D0,
       I1 => D1,
       C => C,
       O0 => S0,
   );

42  C2 : SEL2
   port map(
       I0 => D2,
       I1 => D3,
       C => C,
       O0 => S1,
   );

51  C3 : DFF
   port map(
       D => S0,
       CK => CLK,
       Q => O0,
   );

58  C4 : DFF
   port map(
       D => S1,
       CK => CLK,
       Q => O1,
   );

end structure;

```

```

行
4  VBLKP new_libl_exl_structure(parent)
5  VBLKP parent;
   |
   |  途中省略。
   |
10  h_comp = new_vblk();
   bindeda = bind_libl_exl_structure_sel2(h_comp,
       intsig0+2, intsig0+3, intsig0+6, intsig0+0 );
18  st_component_attr_func(h_comp, "o1", bindeda, h0);

15  h_comp = new_vblk();
   bindeda = bind_libl_exl_structure_sel2(h_comp,
       intsig0+4, intsig0+5, intsig0+6, intsig0+1 );
   st_component_attr_func(h_comp, "c2", bindeda, h0);

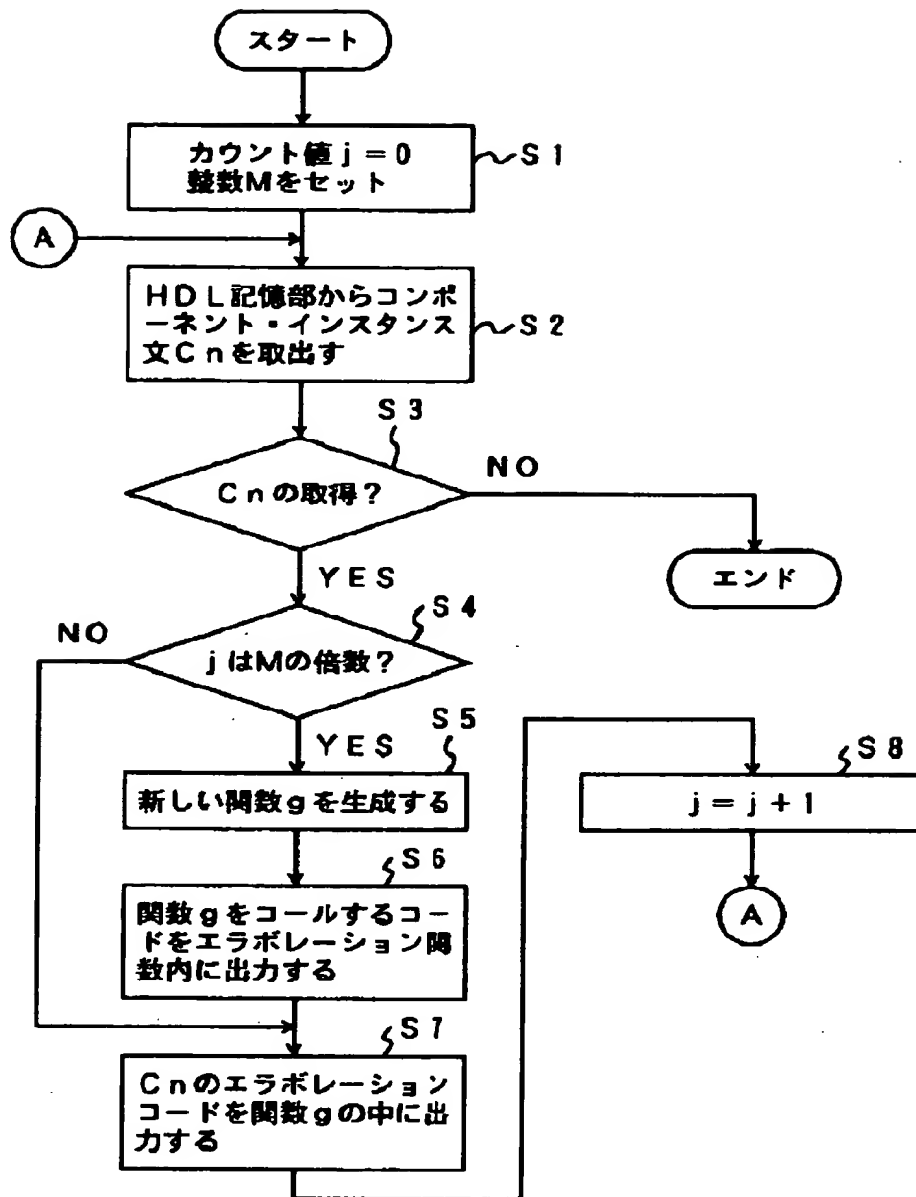
20  h_comp = new_vblk();
   bindeda = bind_libl_exl_structure_dff(h_comp,
       intsig0+0, intsig0+7, intsig0+3);
   st_component_attr_func(h_comp, "c3", bindeda, h0);

25  h_comp = new_vblk();
   bindeda = bind_libl_exl_structure_dff(h_comp,
       intsig0+1, intsig0+7, intsig0+9);
   st_component_attr_func(h_comp, "c4", bindeda, h0);

   return(h0);
}

```

【図3】



【図5】

```

行
3 void new_lib1_ex1_structure_comp0(h0, intsig0)
  VBLKP h0;
  SignalP intsig0;
  {
    VBLKP h_comp;
    VBLKP bindeda;
    extern VBLKP bind_lib1_ex1_structure_sel2();
11    h_comp = new_vblk();
    bindeda = bind_lib1_ex1_structure_sel2(
      h_comp, newpathidx,
      intsig0+2, intsig0+3, intsig0+6, intsig0+0 );
    st_component_attr_func(h_comp, "c1", bindeda, h0);
17    h_comp = new_vblk();
    bindeda = bind_lib1_ex1_structure_sel2(
      h_comp, newpathidx,
      intsig0+4, intsig0+5, intsig0+6, intsig0+1 );
    st_component_attr_func(h_comp, "c2", bindeda, h0);
  }

26 void new_lib1_ex1_structure_comp2(h0, intsig0)
  VBLKP h0;
  SignalP intsig0;
  {
    VBLKP h_comp;
    VBLKP bindeda;
    extern VBLKP bind_lib1_ex1_structure_dff();
34    h_comp = new_vblk();
    bindeda = bind_lib1_ex1_structure_dff(h_comp,
      intsig0+0, intsig0+7, intsig0+8 );
    st_component_attr_func(h_comp, "c3", bindeda, h0);
39    h_comp = new_vblk();
    bindeda = bind_lib1_ex1_structure_dff(h_comp,
      intsig0+1, intsig0+7, intsig0+9 );
    st_component_attr_func(h_comp, "c4", bindeda, h0);
  }

```

【図9】

```

行
4 VBLKP new_lib1_ex1_structure(parent)
5 VBLKP parent;
  {
    途中略

9    new_lib1_ex1_structure_subsig0(intsig0, h0);
10   new_lib1_ex1_structure_subsig8(intsig0, h0);

12   h_comp = new_vblk();
13   bindeda = bind_lib1_ex1_structure_sel2(h_comp,
      intsig0+2, intsig0+3, intsig0+6, intsig0+0);
15   st_component_attr_func(h_comp, "c1", bindeda, h0);

    途中略

    return(h0);
  }

```

【図13】

```

行
4 VBLKP new_lib1_ex1_structure(parent)
5 VBLKP parent;
  {
    途中略

9    new_lib1_ex1_structure_subsig0(intsig0, h0);
10   new_lib1_ex1_structure_subsig8(intsig0, h0);

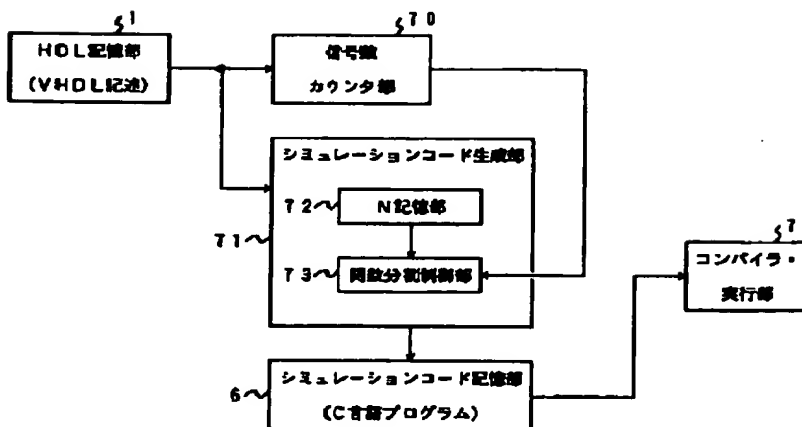
    途中略

15   new_lib1_ex1_structure_comp0(h0, intsig0);
16   new_lib1_ex1_structure_comp2(h0, intsig0);

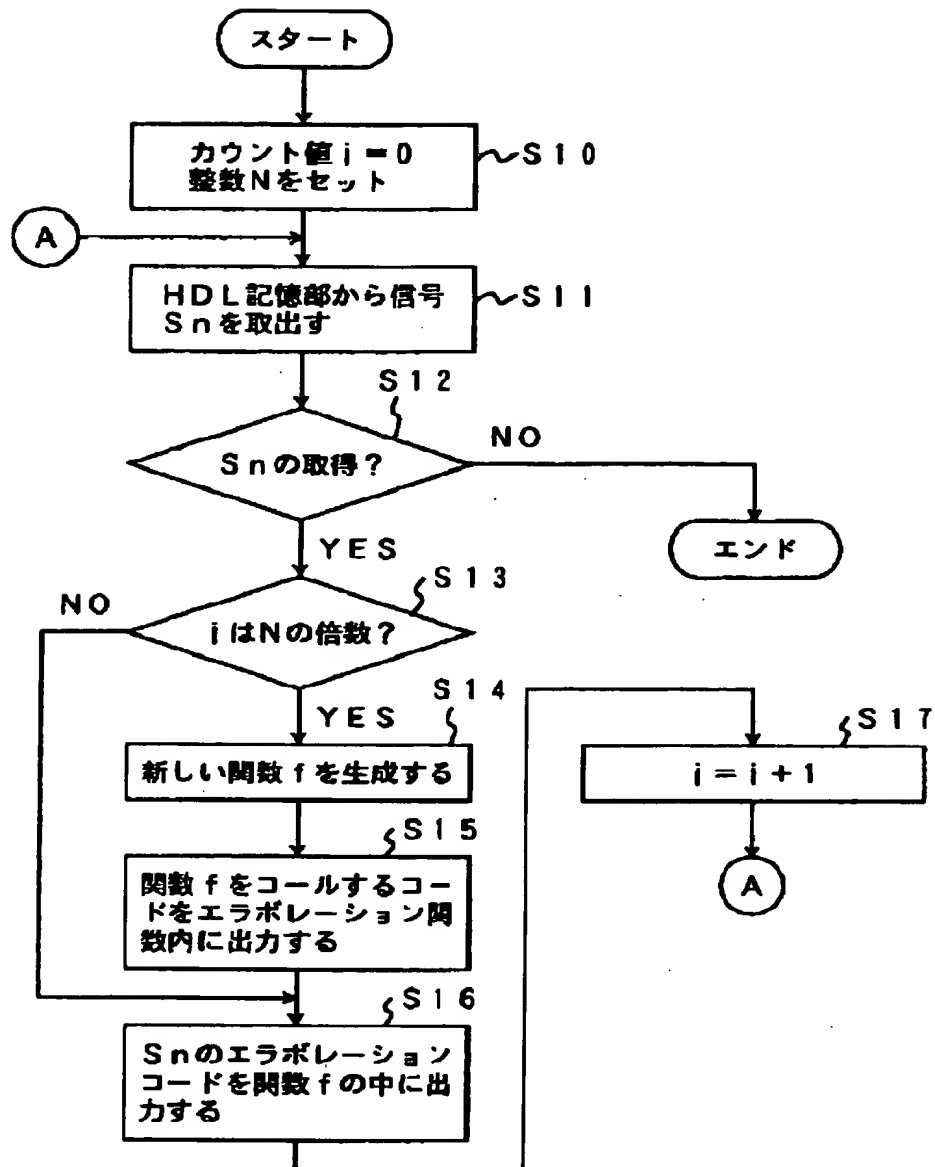
    return(h0);
  }

```

【図7】



【図8】



【図 10】

```

行
3 void new_dia_kurosawa_xrl_structure_subsig0(intsig0, h0)
  SignalP intsig0;
  VBULK h0;
7 {
  st_signal_attr_func(intsig0+0, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+1, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+2, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+3, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+4, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+5, SCALAR_SIGNAL, NULL, h0);
18 void new_dia_kurosawa_xrl_structure_subsig6(intsig0, h0)
  SignalP intsig0;
  VBULK h0;
22 {
  st_signal_attr_func(intsig0+6, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+7, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+8, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+9, SCALAR_SIGNAL, NULL, h0);
  }

```

【図 11】

```

行
4 VBULK new_dia_kurosawa_xrl_structure(parent)
  VBULK parent;
  int pathidx;
  |
  | 途中略。
10 st_signal_attr_func(intsig0+0, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+1, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+2, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+3, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+4, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+5, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+6, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+7, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+8, SCALAR_SIGNAL, NULL, h0);
  st_signal_attr_func(intsig0+9, SCALAR_SIGNAL, NULL, h0);
22 h_comp = new_vblk();
23 bindeda = bind_libl_xrl_structure_sel2(h_comp,
  intsig0+2, intsig0+3, intsig0+8, intsig0+9);
  st_component_attr_func(h_comp, "c1", bindeda, h0);
  |
  | 途中略。
  |
  return(h0);
  |

```

【図 12】

